

Remarks

This responds to the Advisory Action dated January 30, 2008 (the “Advisory Action”) and the Office action dated August 28, 2007 (the “Office Action”). Applicants respectfully request reconsideration of the present application in view of the foregoing amendments and the following remarks. Claims 1-33 are pending in the application. Claims 1-33 are rejected. No claims have been allowed. Claims 1, 19 and 33 are independent. Claims 1, 19, and 33 have been amended.

Cited Art

The Office Action and Advisory Action cite Davidson et al., USPN: 6,083,276 (“Davidson”).

Claim Rejections under 35 USC § 102

The Office Action rejects claims 1-33 under 35 USC 102(b) as being anticipated by Davidson. Applicants respectfully submit the claims are allowable over the cited art. For a 102(b) rejection to be proper, the cited art must show each and every element as set forth in a claim. (See MPEP § 2131.01.) However, the cited art does not describe each and every element. Accordingly, applicants request that all rejections be withdrawn. Claims 1, 19 and 33 are independent.

Claim 1

Claim 1, as amended, recites, in part:

A computer implemented method for producing a data domain for a data structure element of an executable computer program to be used to target efficient testing of behavior of the program during its execution, the executable computer program having been compiled into executable form, the method comprising:

...
receiving a reflection of the executable computer program during execution of the program;
producing the data domain based on the domain configuration information and the reflection of the executable computer program, the data domain representing a limited set of data values to be used as input during execution of the computer program for testing execution of the executable computer program;
targeting testing during execution of the executable computer program to use only values for the data structure element that fall within the data domain; and
determining whether the executable computer program behaves correctly when executing using targeted values ...

[Emphasis added.] For example, the Application describes the motivation for producing data domains to be used during testing:

One aspect of testing and verification of software (regardless of the type of automation) that is particularly challenging is the ability of a testing tool to define a finite domain of data to be used in the testing. For instance, if a program accepts as one of its data inputs the basic data type of Integer that would mean that for a testing tool to exhaustively test the software it would have to conduct the test by applying virtually an infinite number of different integers. However, that is not only costly and time consuming but it also may be meaningless. Thus, a tester may . . . manually limit the domain of each such data member to a selected set of values. For instance, if Age is a field of the type Integer then a meaningful domain for such a data member may be limited to integers ranging from 1-100. . . . Thus, the process of testing can be vastly improved by specifying, or configuring the domain of the various data structure elements related to software programs.

[Application, at page 1, line 30 to page 2, line 11.] The application proceeds to provide additional examples of data domains, and how they contain limited data to be used during testing, starting at page 8:

In the case of methods the domain may be an enumeration of a set of tuples of its parameters. A set of tuples is a collection of values for a set of variables. For example, a one member tuple is represented as (a_1) , an ordered pair is a two member tuple represented as (a_1, a_2) , a three member tuple is represented as (a_1, a_2, a_3) and so on. FIG. 3B illustrates a data domain 325 for the method Run 320 (defined in FIG. 1), which receives the parameters of Age and Weight. Thus, the data domain for the Run method 122 is a set of two member tuples of various combinations of values of Age and Weight. For example, these tuples may be as follows:

$(Age, Weight) = [(10, 50), (12, 55), (45, 167) \dots (20, 150)]$

...

The data domain for a data type is a simple enumeration of the possible values of the data type. For example, in case of a basic type like Integer the domain may be a set of integer values for the Integer type to be used in the program. In this case, the domain may be represented as follows:

$[1, 2, 3, 4, 5, \dots 100]$

[Id. at page 8, line 29 to page 9, line 22; emphasis added.]

The Application also describes the acquisition of the reflection of a program, as obtained from an executable version of the program during execution of the program:

The reflection of a computer program in part comprises the meta data related to the data structure elements used in the program. . . . Developer tools currently available provide for features whereby the reflection of a computer programs can be read to obtain the information related to the data structure elements employed in the programs.

For example, .NET framework by Microsoft® Corporation supports features for allowing an executing program to introspect upon itself to obtain information (meta data) related to its classes or data types, fields, methods, parameters etc. The Java® programming language by Sun® Microsystems allows for similar features. Older programming languages such as C, FORTRAN, and even C++ do not have such features. However, some such languages are supported by the .NET® framework (e.g., C, C++, FORTRAN etc.) and once compiled within the context of the .NET® framework these same reflection features may be used to obtain the reflection of programs coded in all the .NET® supported languages.

[Application, at page 10, line 18 to page 11, line 3; emphasis added.] The Application goes on to illustrate the connection between working with reflections of executable programs, including the ability to receive information about frameworks the programs execute within:

For example, if a program has been compiled using the tools of the .NET.RTM. framework, upon reading a reflection of such a program the domain configuration manager 410 should have access to all the methods of the .NET.RTM. framework libraries and these methods can be used in the expression for configuring the data domain of the program for its testing or verification.

[Application, at page 13, lines 12-16.]

Davidson cannot “receive[e] a reflection of [an] executable computer program,” as recited in claim 1 because Davidson’s actions are performed in order to generate a program, and therefore are done on an incomplete program. Davidson is titled “Creating and configuring component-based applications using a text-based descriptive attribute grammar” Further, the Application describes the general process of Davidson as:

A method for creating and configuring a component-based application through text-based descriptive attribute grammar includes creating a parse tree from an application description file, transforming the parse tree into a plurality of components corresponding to instances of classes in an application framework, and initializing and further processing the components to launch the component-based application.

[Davidson, at Abstract.] Davidson goes on to describe the general process of creating an application according to its techniques:

Referring now to FIG. 2, there is shown a high-level dataflow diagram of the system 100 in accordance with a preferred embodiment of the present invention. The parser 116 receives an application description file (ADF) 202 . . .

The parser 116 parses the ADF 202 to generate an XML parse tree 204. . . The ADF 202 and the parse tree 204 are described in greater detail below with respect to FIG. 3A.

Thereafter, the parse tree 204 is transformed by the element processors 118 into a plurality of uninitialized components 212. In one embodiment, the components 212

are "bean" objects in the JavaBeans application framework 130. The process of mapping elements 306 to components 212 is discussed below in greater detail with reference to FIG. 3B.

After the components 212 are created, the element processors 118 process the uninitialized components 212 to launch the component-based application 214. As will be described in greater detail below with reference to FIGS. 3C and 4D, this is accomplished in one embodiment by initializing each of the components 212 and adding each child component 212 to its parent component 212.

[Davidson, at column 6, line 39 to column 7, line 5.]

In its rejection of the "reflection" language of claim 1, the Office Action refers to the uninitialized components 212 of Figure 2; these are also referred to as Java "beans." [Davidson, at column 6, lines 59-63, quoted above.] As the passage quoted above shows, however, these uninitialized components are created, and then processed to "launch the component-based application 214." Applicants thus respectfully note that these uninitialized components cannot read on "a reflection of [an] executable computer program" as recited in claim 1, as no executable version, nor even a compiled version, of the program yet exists at the time the components are used. In particular, Applicants note the process of Figure 4C, which is cited in the Office Action, which results in *methods being written* for a program being created based on descriptors and uninitialized bean components. [Davidson, at column 25, lines 8-31.] Because methods have yet to even be written after the beans are created, it does not make sense to speak, in Davidson, as there being an "executable computer program" during creation and processing of the beans, and they cannot read upon this language of claim 1.

Applicants further note that Davidson nowhere mentions compilation of code into a executable program. However, even if Davidson were, for the sake of argument, deemed to be compiling the bean objects *into* a executable program, this should not imply that the uninitialized components themselves are a reflection of the program. As discussed above, at the time the components are used, there does not yet exist a completed program, and the program only exists after processing of the components themselves. For at least this reason, the uninitialized bean components of Davidson must exist before a compiled program exists, and cannot, therefore, reflect an executable program.

Davidson's mapping of element attributes to descriptors of bean components also cannot "target[] testing during execution of the executable computer program" or "determin[e] whether the executable computer program behaves correctly when executing using targeted values" as recited in

claim 1 because Davidson's actions are performed in order to generate a program, and therefore are done on an incomplete program, which is therefore not executable. In its rejection of this language of claim 1 the Office Action recites to the passage of columns 25 and 26 referring to the process of Figure 4C. As mentioned above, however, this passage describes a process for creating a program and writing methods for that program. In fact, the process of Figure 4C performs the "Map Element Attributes to Component Properties" process of step 418 of Figure 4B, which, in turn, performs the "Transform Parse Tree Into Uninitialized Components" step 404 of Figure 4A. [See Davidson at column 24, lines 35-41 and at column 21, line 66 to column 22, line 2.] As such, it is performed before the "Process Components to Launch Application" step 406 of Figure 4A, and thus is performed before creation of a program. Thus, it cannot read on "targeting testing during execution of the executable computer program" or "determining whether the executable computer program behaves correctly when executing using targeted values" as recited in claim 1 as these each recite an "executable computer program" which "ha[s] been compiled into executable form."

Davidson does not teach or suggest "determining whether the executable computer program behaves correctly when executing using targeted values falling within the data domain is input" as it does not describe testing of an program during execution. Davidson does not describe testing of programs at any point. In its rejection of claim 1, the Office Action alleges that the "targeting testing" language of claim 1 can be found in the portion of Davidson accompanying Figure 4C, i.e. the passage discussed above which describes the mapping of element attributes to component properties. Applicants first note again that no testing per se is performed here, merely a check to see if mappings exist *before creation of the executable program*. More importantly, Applicants note that this passage cannot teach or suggest the "determining" language of claim 1 because it does not operate on an executing computer program. This is true at least for the fact, discussed above, that at the point of the process of Figure 4C, no program has yet been created. Thus, there is no executing program to test, and the process cannot read upon the quoted language of claim 1.

Applicants also note that, while Applicants respectfully disagree that the term "data domain" carried the scope suggested in the Office Action, the claim has been amended to recite that "the data domain represent[s] a limited set of data values to be used as input for testing an execution of the computer program." Applicants believe this will clarify understanding of the claim and, along with the arguments above, demonstrates that the claim is in condition for allowance.

Finally, Applicants respectfully disagree with the contention, made in the Advisory Action, that the amended claims (as suggested in the Amendment of January 4, 2008) would not render the claims allowable over Davidson. The Advisory Action states, in part:

[I]n the hypothetical case where the proposed changes would be entered, the claim amounts to receiving metainformation and configuration information, to produce a set of ranges (based on compiled form of program such as a CFG) of values serving as input to verify if the program when simulated with such a set of values, would not yield errors. In that light, the claimed invention is not distinguishing from code verification such as the tree-based code testing/debugging by Davidson, among others.

[Advisory Action, at page 2.] While Applicants do not necessarily agree with the Advisory Action's interpretation of the claims, Applicants point out that, *even if this interpretation were correct, Davidson would still not read on the claimed invention.* David does not teach debugging or code testing, but, as discussed above, teaches the generation of a computer program and, as part of that generation, Davidson checks to see if mappings exist. Applicants contend that this does not teach or suggest "testing." Additionally, even if Davidson show "tree-based code testing/debugging," Applicants note that operations on trees do not show testing, or any other operation, on an "executable computer program" as recited in the claim.

For at least these reasons, the cited passages of Davidson cannot describe the above-quoted language of claim 1. Furthermore, Applicants do not find further such disclosure elsewhere in Davidson. Applicants thus request that the rejection of claim 1 be withdrawn and that the claim be allowed.

Claim 19

Claim 19 recites:

using a reflection of the executable computer program produced during execution of the program to produce the data domain for the data structure element according to the domain configuration information, the data domain representing a limited set of data values to be used as input for testing execution of the executable computer program; and

controlling testing during execution of the executable computer program to use only values for the data structure element that fall within the data domain.

[Emphasis added.] In its rejection of claim 19, the Office Action cites to the same passages of Davidson as it does in its rejection of claim 1. Thus, for at least the reasons discussed above with

regard to claim 1, Davidson does not describe each and every element of claim 19. Applicants thus request that the rejection of claim 19 be withdrawn and that the claim be allowed.

Claim 33

Claim 33 recites:

means for reading, on the computer apparatus, *a reflection of the executable computer program during execution of the program;*

means for processing, on the computer apparatus, the domain configuration information and the reflection to produce and output the data domains corresponding to the data structure elements, *the data domains representing a limited set of data values to be used as input for testing execution of the computer program;* and

means for limiting testing during execution of the executable computer program to use only values for the data structure element that fall within the data domain.

[Emphasis added.] In its rejection of claim 33, the Office Action refers directly to its rejection of claim

1. Thus, for at least the reasons discussed above with regard to claim 1, Davidson does not describe each and every element of claim 33. Applicants thus request that the rejection of claim 33 be withdrawn and that the claim be allowed.

Dependent Claims

The Office Action additionally rejects claims 2-18 and 20-32 under 35 USC § 102(b) as being anticipated by Davidson. Each of these claims, however, depends from either independent claim 1 or 19 and recites patentably distinct subject matter not described by the cited art. However, Applicants do not belabor the language of the individual claims in the interest of brevity but instead note that for at least the reasons given above with respect to the independent claims, the dependent claims are allowable. Applicants request that the rejections of claims 16 and 19 be withdrawn and that the claims be allowed.

Interview Request

If the claims are not found by the Examiner to be allowable, the Examiner is requested to call the undersigned attorney to set up an interview to discuss this application.


Conclusion

The claims in their present form should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By 

Stephen A. Wight
Registration No. 37,759